DOI: 10.2298/CSIS120716045A

Mobile Agent Group Communication Ensuring Reliability and Totally-ordered Delivery

Jinho Ahn[1]

Department of Computer Science, Kyonggi University
Iuidong, Yeongtonggu, Suwon-si Gyeonggi-do 443-760, Republic of Korea
jhahn@kgu.ac.kr

Abstract. In this paper, an atomic mobile agent group communication protocol is designed to have the following beneficial features. The protocol enables each agent to adaptively choose a small number of forwarders among its visiting nodes based on its decision. Also, it replicates paths on which messages should be transmitted to their targeting mobile agents in an effective manner. The mobile agent group location cache each sending agent keeps in our protocol can considerably speed up message delivery to a group of agents and lower message forwarding load imposing on forwarders. Furthermore, it allows messages destined to a group of agents to be reliably delivered to all surviving agents in the same order despite their mobility and F forwarders' failures unlike the other existing ones. Simulation results indicate that our protocol considerably performs better than the existing one in terms of message delivery time and location information management overhead.

Keywords: mobile agent, group communication, scalability, reliability, totally-ordered delivery.

1. Introduction

Mobile agent is an autonomous and independent software program to satisfy the corresponding user's goal on behalf of the user while visiting various target nodes through a network [1], [4], [5], [8], [16]. This mobile agent technology has several advantages such as reduction of network traffic, overcoming of network delay, enabling asynchronous execution and enhancement of dynamic adaptability [4], [5], [8]. Thanks to these desirable features, this technology is very widely used in distributed systems.

However, as the size of these fields is rapidly increasing, several research issues related to the mobile agent technology such as communication, security, dynamic adaptation, etc., should be reconsidered to be suitable for their scale. Among them, it is the most important issue to enhance the performance of the agent communication in Internet-scale infrastructures. For this purpose, some effective and efficient inter-agent communication protocols are required in distributed agent-based systems.

Most of existing mobile agent communication protocols [3], [9], [10], [11], [12] are focusing on the point-to-point based communication issues for scalable

and reliable message delivery to mobile agents despite their migrations. However, as community-based computing has prevailed as a general model of current and future generation computing, group-based communication is becoming an essential building block for this computing. Thus, the inter-agent communication protocols for mobile agents also require the third feature, agent group communication. Furthermore, as atomic group communication semantics providing both agent group communication reliability and totally ordered message delivery is becoming an essential demand for many P2P-based group applications such as P2P file replication for high assurance, multimedia games, video conferencing, message-based social networking, etc. [6], [15], it should been considered in this literature. In here, totally ordered message delivery ensures that messages sent to a set of group members are delivered by all those members in the same order. But, to the best of our knowledge, among the existing protocols, there is no one addressing the important three issues harmoniously due to their respective limitations. Apart from these previous works, several multi-cast protocols considering node mobility being developed in Mobile IP fields [15] may be applied for mobile agent group communications. However, as they are designed for supporting the transparent sub-net change of a mobile node, they are not appropriate for handling the migration of a group of software agents from one computer to another because of their different characteristics.

In this paper, we propose an atomic mobile agent group communication protocol to achieve all the requirements mentioned above. This protocol improves scalability by enabling each mobile agent to choose only a few among its visiting nodes as agent location manager depending on its preferred policies such as location updating and message delivery costs, security, network latency and topology, inter-agent communication patterns, etc.. Second, to guarantee agent communication reliability despite agent location managers' failures, it allows each mobile agent's location information to be replicated in an effective way to preserve its scalability to a maximum. Also, it has messages destined to an agent group to be reliably delivered to its surviving group members in the same order. Lastly, each sending agent's agent group location cache significantly allows message delivery time to the targeted mobile agents to be shortened and message forwarding load imposing on agent location managers to decrease.

The rest of the paper is organized as follows. In section 2, we describe the distributed agent-based system model assumed and in section 3, review related works. Sections 4 and 5 present an atomic mobile agent group communication protocol and prove its correctness. In sections 6 and 7, we show simulation results and conclude this paper.

2. System Model

This paper considers an asynchronous distributed agent based system where there is no global memory, no global clock and no bound on message delay. The system consists of a set of agent service nodes. Each service node supports an environment in which agents can operate safely and securely, and

provides a uniform set of services through which visiting agents can access its local resources in a limited way regardless of their locations. An agent is initially created on a service node, called $home\ node$ of the agent, and is given a unique identifier within the node. So, each agent can be identified as a globally unique object in the system by using the combination of its local identifier and the identifier of its home node. When an agent migrates in the system, its code and state information are captured and then transferred to the next node. After arriving at the node, the mobile agent resumes and performs its task, if needed, by interacting with other agents. In order to perform an assigned task on behalf of a user, a mobile agent AID executes on a sequence of $l(l > 1)$ service nodes according to its itinerary, I_{AID} =[$DN_{home}, DN_1, \cdots, DN_{(l-1)}$], which may be statically determined before the mobile agent is created on its home node or dynamically while progressing its execution. It is assumed that communication channels support standard asynchronous message passing with the reliable FIFO ordering property and are immune to partitioning. Mobile agents can migrate and messages be passed along these channels. Finally, we assume that service nodes and mobile agents crash based on the fail-stop model, in which they lose contents in their volatile memories and stop their executions [14].

3. Related Work

Broadcast-based protocol [12] guarantees transparent and reliable inter-agent communication and can also provide multicast communication to a set of agents. But, to locate the message destination, the protocol has to contact every visiting node in the network. Thus, its large traffic overhead makes broadcasts impractical in large-scale distributed agent systems.

In home-based protocol [9] inspired by Mobile IP [13], every mobile agent has a home node and should register its current location with the home node whenever it moves. Thus, when some messages are sent to a mobile agent currently located at a foreign node, the messages are first directed to its home node, which forwards them to the agent. This protocol is simple to implement and results in little mobile agent locating overhead. However, it is unsuitable for highly mobile agents in distributed agent based systems because every agent location updating and message delivery are all performed around the home node, which introduces centralization. Additionally, in the distributed agent-based systems, the home node may be disconnected from the network.

Forwarding pointer-based protocol [10] forces each node on a mobile agent's movement path to keep a forwarding pointer to the next node on the path. Thus, if a message is delivered to an agent not being at the home node, the message must traverse a list of forwarders. Thus, this protocol can avoid performance bottlenecks of the global infrastructure, and therefore improve its scalability, particularly in large-scale distributed agent-based systems, compared with the home based one. Additionally, even if a home node is disconnected from the rest of the network, the forwarding pointer based protocol allows agents registering with the node to communicate with other agents. However, as highly mobile

agents leads to the length of their chains of pointers being rapidly increasing, its message forwarding overhead may be significantly larger. Furthermore, the number of forwarding pointers each service node needs to keep on its storage may exponentially increase if a large number of mobile agents are running in the systems. In a previous work [10], a type of update message called $inform$ message was introduced to include an agent's current location for shortening the length of trails of forwarding pointers. In this case, a node that receives the message is allowed to update its table if the received information is more recent than the one it had. However, it introduces no concrete and efficient solutions for this purpose, for example, when update messages should be sent, and which node they should be sent to. To consider failures of forwarders, a fault-tolerant directory service for mobile agents using redundancy of forwarding pointers [11] was proposed.

Mailbox-based protocol [3] was proposed to provide location-independent reliable message delivery. It allows messages to be forwarded at most once before they are delivered to their receiving agents. Also, the movement of agents can be separated from that of their mailboxes by determining autonomously whether each mailbox is migrated to its owner agent. However, uncertainty of message delivery to mailboxes may result in useless early pollings. On the other hand, even if urgent messages are forwarded to a mailbox on time, they can be delivered to its corresponding agent very late depending on the agent's polling time. Moreover, whenever each mailbox moves, its new location information should be broadcasted to every node which the mailbox has visited. This may incur high traffic overhead if assuming most agents are highly mobile.

All the above stated protocols haven't ever consider reliable agent group communication with atomic message delivery order.

4. The Proposed Protocol

4.1. Location Updating Operation

First of all, before describing our proposed protocol, let us define two important terms, $forwarder$ and $locator$, according to the role of location managers. Forwarder of an agent is a directory service node that maintains a forwarding pointer of the agent on its storage. Thus, there may be the various number of forwarding nodes of each agent in the system according to which agent communication protocol is used. Locator of an agent is a special forwarder managing the identifier of the service node that the agent is currently running on. Assuming every node is failure-free, our protocol requires only one locator for each mobile agent to address agent mobility. But, if the protocol intends to tolerate up to $F(F \geq 1)$ node crash failures, $(F+1)$ locaters of each mobile agent should exist. Our location updating algorithm is designed to guarantee reliable delivery to mobile agents despite both agent mobility and F failures of forwarders as follows. In this algorithm, every mobile agent α should keep the following information.

• $LCInfo_\alpha$: It is the location information of agent α consisting of two fields, $l\_locators$ and $agent\_t$. $l\_locators$ is a set of identifiers of agent α's live locators. $agent\_t$ is the timestamp associated with agent α. Its value increments by one whenever the agent moves to a new node. Thus, when agent α migrates to a new node, it should inform only locators of the agent in $l\_locators$ of both its new location and $agent\_t$ so that the locaters can locate the agent.

Each forwarder N_i has to maintain the following agent location information on its storage.

• $NextFwdrs_i$: It is a vector which maintains the location information of every mobile agent which is not currently running on N_i, but which N_i is a forwarder of. Its element consists of six fields, $agent\_id$, $l\_nextfwdrs$, c_node, $agent\_t$, $manage\_f$, $migrate\_f$ and msg_Q. $l\_nextfwdrs$ is a set of identifiers of the nodes which N_i thinks are the locators of agent $agent\_id$. If node N_i is the agent's current locator, this field is set to a empty set. In this case, the third field c_node contains the identifier of the node where the agent is currently running. Otherwise, the field c_node's value isn't used for message forwarding. $agent\_t$ is the timestamp associated with the agent when being located at the latest among the nodes in $l\_nextfwdrs$. It is used for avoiding updating recent location information by older information [10]. $manage\_f$ is a bit flag indicating if N_i is a locator of agent $agent\_id$ or not. In the first case, its value is set to $true$ and otherwise, $false$. $migrate\_f$ is a bit flag designating if the agent is currently moving to another node(=$true$) or not(=$false$). msg_Q is a message queue for buffering all the messages destined to agent AID on its migration. Its element consists of two fields, a message m and the source node ID of the message, SID.

Then, we assume that the value of F is 1 in all examples shown later for explaining. First, let us see how to perform failure-free operations of the algorithm using Fig. 1. In this figure, each circle stands for a service node and an agent α is going to travel from its home to another 12 nodes in order. Also, each gray shaded circle stands for a forwarder of the agent. Initially, when the agent is created, its home node becomes the first forwarder and locator of the agent. Unlike the previous forwarding pointer-based protocols [10], [11], our protocol allows each agent to select only a few among its visiting nodes as its forwarders depending on its decision. In this example, when the agent is migrating from node N_1 to node N_2, it forces N_1 to be its next forwarder. In this case, both N_{home} and N_1 become agent α's two locators. Afterwards in our protocol, agent α should register its location with its current locators N_{home} and N_1 on every migration until it decides its new forwarder. When the agent is going to N_6, it determines that N_5 is a proper node as its next forwarder. Then, N_5 informs N_{home} that both N_5 and N_1 play the role of agent α's locator from now. Thus, the current location of the agent will have been managed by N_5 and N_1 until the agent selects N_{10} as its next forwarder. Afterwards, N_{10} and N_5 become the locators of the agent.

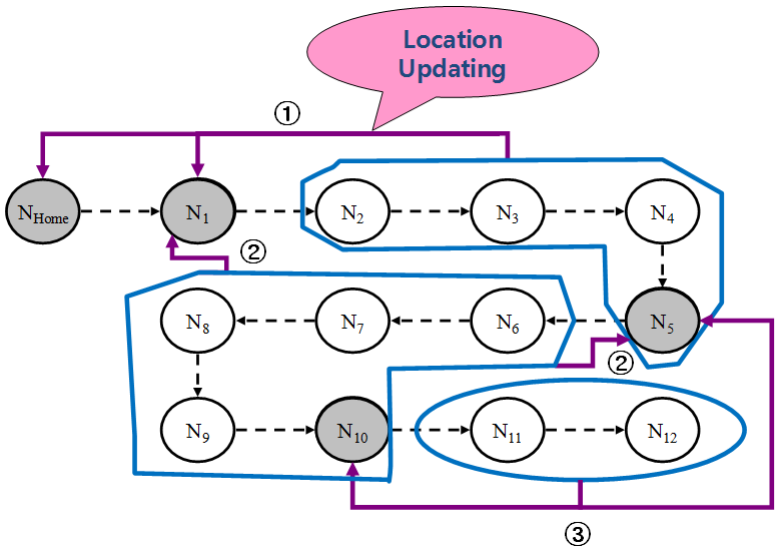

Fig. 1. An illustration of our location updating procedure on locators

In this way, our algorithm can control adaptively the tradeoff between location updating and message delivery costs depending on agent's preferences.

4.2. Atomic Message Delivery Operation to Mobile Agent Groups

To ensure messages to a mobile agent group be reliably delivered to all the live members of the group in the same order, our protocol requires the following data structures for sending or receiving agents. First, every agent α in a group g should keep the following membership information.

• $GMInfo_{g,\alpha}$: a table for saving location information of the other agents which are members of agent group g. Its element is a tuple $(agent\_id, l\_fwdrs)$. l_fwrdrs is a set of identifiers of every other agent $agent\_id$'s most recent forwarders. Initially, $agent\_id$'s most recent forwarder is set to its home node in l_fwdrs.

Each sending agent β has to maintain the following mobile agent location cache.

• $GMCache_{\beta}$: It is a table which temporarily keeps location information of each mobile agent which agent β communicates with. Its element consists of four fields, $group\_id$, $agent\_id$, l_fwdrs and $agent\_t$. l_fwdrs is a set of identifiers of the nodes which agent β guesses are locators of agent $agent\_id$ in group $group\_id$. Thus, to send messages to agent $agent\_id$, agent β forwards them to the latest among $agent\_id$'s live locaters in l_fwdrs. If there is no live locater in

l_fwdrs, the messages are sent to the home node of agent $agent\_id$. $agent\_t$ is the timestamp assigned to agent $agent\_id$ when the agent registered with the latest among all locators in l_fwdrs.

In our protocol, among mobile agents in a group, a group leader is selected and determines the order of each received message. If a sending agent β is attempting to transmit a message m to a group g, it broadcasts the message to the latest forwarder γ of every member agent α of group g by consulting $GMCache_{\beta}$. Then, message m can be routed from γ to a live locator of agent α and then arrive at the node where agent α is currently running. At this time, delivering the message to agent α's application should be delayed until the message order is determined in the group. When the group leader receives the message, it assigns a sequence number to the message and informs the other members of the number based on its $GMInfo_{g,leader}$. If a member α detects missing an message sent to its group occurring from its sender's failure when the agent has obtained the delivery order of the message from its group leader, it has to solicit the leader to retransmit the message. Afterwards, each member agent can deliver the message to the corresponding application in order. In this case, the agent must delay executing all the output commit actions[7] depending on the messages delivered until the group agreement on their delivery orders is completed. Additionally, it should keep both the identifier and the delivery sequence number of the message with itself into its buffer and acknowledges the number assignment to the group leader. Then, the leader notifies its live members of the group agreement on the message delivery order. After receiving the global commitment from the leader, each non-faulty member can purge the information for the multicast message from its buffer.

Let us illustrate how our atomic message delivery algorithm based on the fault-tolerant location updating algorithm mentioned in section 4.1 satisfies our goal using three examples. In Fig. 2, there is a mobile agent group $gp1$ consisting of two agents MA_1^1 and MA_1^2, which move from their home nodes DN_A and DN_B to new nodes DN_{A+i} and DN_{B+2} according to their itineraries. In this case, agents MA_1^1's and MA_1^2's first locators become DN_A and DN_B respectively, which manage the current location information of the two agents in $NextFwdrs_A$ and $NextFwdrs_B$. Also, suppose that the first agent MA_1^1 is the group $gp1$'s leader. Each agent has to maintain the location information of the other agent members in its group like $GMInfo_{1,1}$ and $GMInfo_{1,2}$. Afterwards, a sending agent MA_{β} on node DN_S attempts to send a message m_1 to group $gp1$. In our protocol, each sending agent keeps an agent group location information cache to accelerate message delivery to its frequently communicating agent or agent group. The cache is initially set to record each agent's home node information in its corresponding element like $GMCache_{\beta}$. Thus, in this case, as the sending agent has no short-cut information to agent members of group $gp1$, it sends the message to their respective home nodes DN_A and DN_B, which forward the message to their corresponding agents by looking up $NextFwdrs_A$ and $NextFwdrs_B$. When receiving the message, each agent

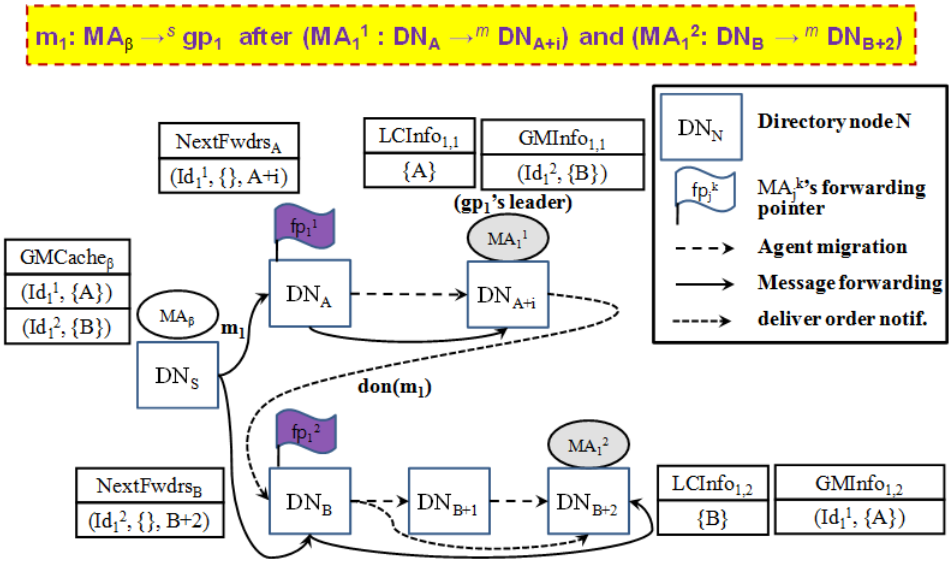

Fig. 2. In case agent MA_β sends a message m_1 to group gp_1 after two agents MA_1^1 and MA_1^2 in gp_1 moved to nodes DN_{A+i} and DN_{B+2} respectively

shouldn't deliver it to the corresponding application until its group leader notifies itself of the global sequence number of the message to ensure the atomic message delivery condition. So, the group leader MA_1^1 increases the global sequence number by one and informs the other agent members of the number by sending a control message $don(m_1)$ with it based on its membership information $GMInfo_{1,1}$. Then, the control message is transmitted to agent MA_1^2 via its current locator DN_B allowing the message m_1 to be delivered to the application. Hereafter, although the following procedure isn't drawn in this figure, the global agreement on the message delivery order should be reached and then notified its live members.

Fig. 3 and 4 show an example that the second agent MA_1^2 migrates to new nodes DN_{B+3} and DN_{B+4} in order before the sending agent MA_β are going to send another message m_2 to group $gp1$. In this example, as MA_1^2 moves from DN_{B+3} to DN_{B+4}, it decides DN_{B+3} is a proper node as its new locator. From this time, MA_1^2 will directly be located by both DN_B and DN_{B+3}. Then, the message m_2 can be forwarded to all the agent members of group $gp1$ in the same way as mentioned above. But, in Fig. 3, as MA_1^2 can recognize the sending agent didn't have the new locator's information, it informs MA_β of the information by sending a message $CUpdate$ including its two locators' information for considering locator's failure. In this case, MA_β updates the second agent's locators information on its location cache $GMCache_\beta$. Afterwards, when the sending agent are sending other messages to the group $gp1$, the mes-

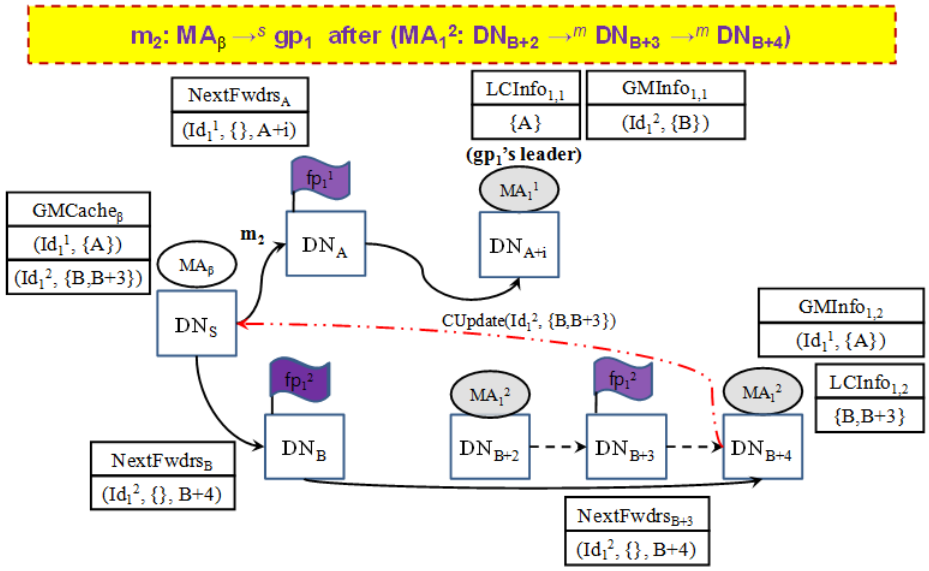

Fig. 3. In case MA_β sends a message m_2 to gp_1 after MA_1^2 has migrated to DN_{B+3} and DN_{B+4} in order

sages can be forwarded directly to the new locator closer to the second agent by consulting the cache, causing message delivery to be accelerated. When MA_1^2 receives a control message $don(m_2)$ from its group leader MA_1^1 like in Fig. 4, it can update the leader on its new locator assignment when returning an acknowledgement about the message delivery number to the leader.

Fig. 5 illustrates how the group leader MA_1^1 updates sending agents like MA_β and its own agent members like MA_1^2 on its new locator selection during its migration. In this example, DN_{A+i} becomes the leader's second locator. In this case, as a new message m_4 is first forwarded by DN_A to MA_1^1, MA_1^1 notifies MA_β of its two locators information on a control message $CUpdate$. Also, when MA_1^1 informs MA_1^2 of m_4's delivery number, its two locators information in message $don(m_4)$ can be reflected on MA_1^2's membership table $GMInfo_{1,2}$.

4.3. Recovery Operation

In mobile agent group communication, there are two kinds of mobile agent failures, general member's and group leader's failures. First, if some agent members except for their group leader crash, their identifiers and locators information have only to be removed from the other surviving agent members' membership table $GMInfo$ in our protocol. However, if the group leader's failure is detected, the following complicated procedure should be performed. Among the other live agent members, a new group leader is elected that has delivered the largest

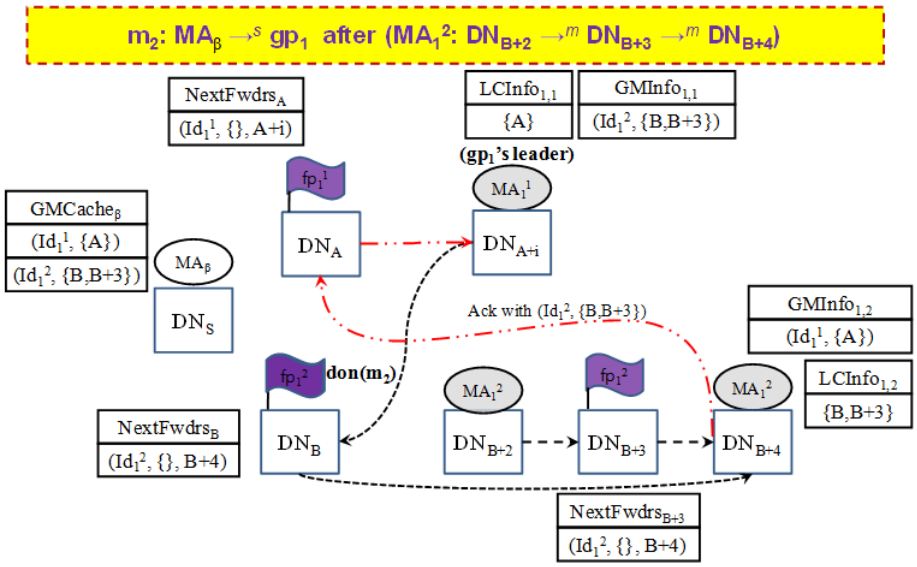

Fig. 4. In case MA_β sends a message m_2 to gp_1 after MA_1^2 has migrated to DN_{B+3} and DN_{B+4} in order(*continued*)

number of messages destined to their group. Then, the leader should force the other live agent members to receive the global delivery sequence numbers of all the messages which it has delivered, but they haven't yet, and deliver the messages in the same order. Afterwards, the following acknowledgement and global commitment procedures mentioned in section 4.2 are performed.

5. Correctness

This section proves liveness and safety of our proposed mobile agent group communication protocol using 5 lemmas.

5.1. Liveness

This section shows our protocol ensure validity and agreement properties in order.

Definition 1. $MsgPassingRouteToMA(msg, AID_T, AID_S)$ *is a sequence* $< DN_i, DN_{i+1}, ..., DN_n >$ *of forwarders of the target mobile agent* AID_T *constituting a route with no cycle for transmitting a message* msg *from the sending mobile agent* AID_S *to* DN_n *in our protocol. In here, the last forwarder* DN_n *is the only live locator of agent* AID_T *in* $MsgPassingRouteToMA(msg, AID_T, AID_S)$ *where its immediate predecessor* DN_{n-1} *has chosen to forward* msg.

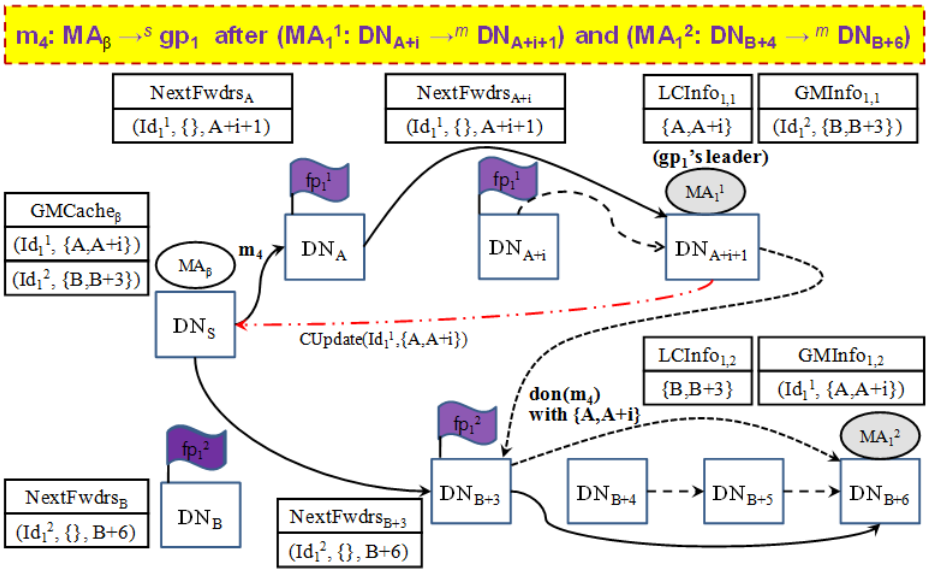

Fig. 5. In case MA_β sends a message m_4 to gp_1 after MA_1^1 and MA_1^2 have visited DN_{A+i+1} and DN_{B+6} respectively

For any node pair, DN_α and DN_β, in the route ($i \leq \alpha < \beta \leq n$), agent AID_T has visited DN_α before DN_β.

Lemma 1. *If more than F forwarders for any target mobile agent AID_T have been created in our protocol, $MsgPassingRouteToMA(msg, AID_T, AID_S)$ is always made even in case of F failures of its forwarders.*

Proof. We prove this lemma by contradiction. Assume that our protocol may not make a route $MsgPassingRouteToMA(msg, AID_T, AID_S)$ if F forwarders or fewer crash. This assumption means there exists one forwarder DN_ϵ, not locator, that cannot progress sending msg to its immediate successor from AID_S any longer. However, according to our protocol, the second element of $NextFwdrs_{DN_\epsilon}$, $l\_nextfwdrs$, contains identifiers of its F+1 immediate successors. So, even if, among them, F successors fail, DN_ϵ can forward msg to the only live one $DN_{\epsilon+1}$. As no more failures of forwarders will occur later, msg can be eventually delivered to the last forwarder DN_n from $DN_{\epsilon+1}$.
Therefore, the route $MsgPassingRouteToMA(msg, AID_T, AID_S)$ made by our protocol exists even in case of F failures. This contradicts the hypothesis. □

Lemma 2. *If an agent δ sends a message m to any member μ of mobile agent group gp_i in our protocol, the message is eventually delivered to the member regardless of its movements.*

Proof. Suppose that the message m should be delivered to the agent μ, and the sender agent δ and the receiver agent μ are currently running on nodes DN_α and DN_β respectively. When DN_α attempts to send the message to agent μ, DN_α first looks up the address of the most recent forwarder for μ from its agent location cache $GMCache_\alpha$. If there is the address in the cache, DN_α sends the message to this address. Otherwise, it obtains the address of μ's home node from μ's identifier and then sends the message to the address. In both cases, suppose the forwarder is denoted by DN_γ. The proof proceeds by induction on the number of all the forwarders $\in MsgPassingRouteToMA(msg, \mu, \delta)$, denoted by $|MsgPassingRouteToMA(msg, \mu, \delta)|$.

[Base case]

In this case, DN_γ is the locator of agent μ. Therefore, the following two cases should be considered.

Case 1: $\gamma = \beta$.

In this case, DN_β can find μ's location information e from a list of agents running on DN_β. There are two subcases considered.

Case 1.1: $e.migrate\_f$ = 0.

In this case, the message msg is trivially delivered to μ because μ is currently running on node DN_β.

Case 1.2: $e.migrate\_f$ = 1.

In this case, agent μ attempts to move to another node DN_τ. Thus, the message is saved on μ's message queue $e.msg\_Q$ in $NextFwdrs_{DN_\gamma}$ until the movement process is completed. After agent μ has migrated to DN_τ, the message can be correctly delivered to μ from its queue on DN_γ when the agent resumes its execution on the node DN_τ.

Case 2: $\gamma < \beta$.

In this case, DN_γ looks up μ's location information e from $NextFwdrs_\gamma$. Then, it checks the value of a flag variable $e.migrate\_f$. There are two subcases according to the value.

Case 2.1: $e.migrate\_f = 0$.

In this case, the message is sent directly to DN_β. Then, the subsequent procedure is performed like in case 1.1. Therefore, μ can correctly receive the message.

Case 2.2: $e.migrate\_f = 1$.

In this case, agent μ is migrating to another node DN_τ. Thus, the message is buffered into μ's message queue $e.msg\_Q$ in $NextFwdrs_\gamma$ until the movement process is completed. After the migration, the value of a flag variable $e.migrate\_f$ changes to 0 and then the message recorded in the queue is forwarded to DN_τ. Therefore, the message can be correctly delivered to μ.

[Induction hypothesis]

We assume that the lemma is true for the message msg in case that $|MsgPassingRouteToMA(msg, \mu, \delta)| = k$.

[Induction step]

After the message has been safely routed to the k-th forwarder DN_ω by induction hypothesis, DN_ω retrieves μ's location information e from $NextFwdrs_\omega$.

Then, it forwards the message to the (k+1)-th forwarder DN_ϵ, which is the locator of agent μ. The following case is similar to the base case mentioned above. Therefore, the message can be correctly delivered to μ.
By induction, our protocol enables the message m sent from the sender δ to be correctly delivered to any member μ of the agent group gp_i despite its movements. □

Lemma 3. *Our protocol terminates within a finite time.*

Proof. There are two cases there may occur blocking in our protocol.
Case 1: agents migrate to another nodes or some of forwarders fail.
For this case, lemmas 1 and 2 proved that there is no blocking involved.
Case 2: the group leader l_{cur} of a mobile agent group gp_i fails.
In this case, a new group leader l_{new} having the most recent message m_{last} whose delivery order is assigned by l_{cur} is elected among the other group members and takes over l_{cur}'s role. Assume $lval$ is the value of the delivery order of m_{last} that has been delivered by l_{new} before l_{cur}'s failure. For ensuring liveness of our recovery procedure, it should be shown that every live group member can deliver and acknowledge all the messages whose receipt order is less than or equal to $lval$ after the procedure completed. According to the recovery procedure, all the messages for which global commitment procedures terminated have already been purged from every group member's buffer. The remaining messages with the delivery order up to $lval$ and their orders must be available in at least l_{new}'s buffer because they are unable to be removed until the global group agreement on their corresponding delivery orders has been completed. Thus, our recovery procedure enables every live group member to deliver them in order.
Therefore, our protocol terminates within a finite time. □

5.2. Safety

This section shows our protocol ensure atomicity and total order properties in order.

Lemma 4. *If a member μ in a mobile agent group gp_i delivers a message m and begins executing any output commit action depending on m, m is delivered by every live group member despite μ's future failures.*

Proof. We prove this lemma by contradiction. Assume that our protocol disables some agent members from delivering m even if μ starts performing any output commit action occurring after having delivered m before μ's failure. This assumption means there are one or more live agents that may not deliver m after the global group agreement on m's delivery order has been completed. However, the agreement finalization should occur only after every live agent in group gp_i including μ has received m's delivery sequence order number from the group leader and delivered it when our protocol executed as explained in section 4.2. This contradicts the hypothesis. □

Lemma 5. *All live agent members of a mobile agent group deliver each message destined to the group in the same order.*

Proof. We prove this lemma by contradiction. Assume that our protocol may not satisfy totally-ordered delivery condition in some cases. This assumption means there are one or more messages, denoted by $DISORDERED$, that live members of a mobile agent group gp_i has delivered in different orders in our protocol. There are two cases to be considered.
Case 1: the group leader l_{cur} of gp_i is alive.
In this case, no message in $DISORDERED$ may exist because every message destined to gp_i is always ordered only by the group leader during failure-free operation.
Case 2: the group leader l_{cur} of gp_i fails.
In this case, a new group leader l_{new} having the most recent message whose delivery order is assigned by l_{cur} is elected among the other group members and takes over l_{cur}'s role. Suppose that $acked$ is the value of the last message to gp_i delivered and acknowledged by all the group members including l_{new} before l_{cur}'s failure. Thus, the order of every message whose delivery order is less than or equal to $acked$ is uniquely assigned by l_{cur}. Secondly, assume $nacked$ is the value of the delivery order of the most recent message that has been delivered by l_{new}, but not acknowledged by all the group members before l_{new} becomes the group leader. All the messages with the delivery order up to $nacked$ are sequenced only by l_{cur} and can be recovered by l_{new} like in lemma 3. Additionally, no message with the delivery order larger than $nacked$ might have been delivered by any live group member. Thereafter, l_{new} defines the delivery order of all the messages whose order is greater than $nacked$.
Therefore, our protocol can guarantee totally-ordered delivery semantics in all the cases. This contradicts the hypothesis. □

6. Simulation

Hereafter, our protocol's effectiveness will be analyzed with respect to message delivery time and location management overhead using a discrete-event simulation language [2]. The counterpart evaluated with ours is the representative faut-tolerant forwarding pointer-based protocol [11], having no agent group support functionality like the other existing ones and sending a message destined to an agent group to each member of the group using its uni-cast primitive. For fair comparisons, this protocol is modified to be capable of ensuring totally-ordered and reliable message delivery like ours. The two compared protocols are each abbreviated by MAGCP(Mobile Agent Group Communication Protocol) and MFPP(Moreau's Forwarding Pointer-based Protocol) [11] in order. The number of node failures, F, they should both tolerate is set to 1, i.e., two locators of each agent should exist. For this evaluation, two following performance indices should be explained. The one is LTMD$_{time}$, the average latency of forwarding a message destined to a mobile agent group from a sender until the

message is actually delivered to all group members. The other is LInfo$_{no}$, the average number of agent location entries that each service node keeps on its storage. A simulated system is composed of 100 nodes each associated with a coordinate(x, y). They are all interconnected with each other. Any two adjacent nodes are connected with a LAN link having a bandwidth of 100 Mbps. For simplicity of this simulation, we assume that both bandwidth and propagation delay between any pair of nodes are proportional to their distance. In our simulation environment, there are several important simulation parameters as follows. The first is Set$_{snd}$, the set of nodes being capable of sending messages to a mobile agent group. Nodes in the set are uniformly distributed along the entire network. The second is LChg$_{thrd}$, the threshold which is required when a mobile agent decides whether the current locator of the agent should be changed. If a randomly generated probability variable δ for an agent ma is greater than its threshold LChg$_{thrd}$ when the agent attempts to migrate to another node, the next node becomes the locator of the agent. In this simulation, it is assumed that the threshold LChg$_{thrd}$ is determined when its agent is created and isn't changed during the entire life cycle of its agent any more. The third is Stay$_{time}$, the mean time elapsed when a mobile agent stays at a node for performing its task, following an exponential distribution. The last simulation parameter is MAGroupSize$_{no}$, the total number of agents created in the system and joining a mobile agent group. Also, the size of each application message transmitted between any two agents ranges from 512 bytes to 1 kbytes and the size of each control message for the inter-agent communication is 128 bytes. In addition, messages to the agent group are sent to the network with a interval following an exponential distribution with a mean T_{ms}=100ms. All experimental results shown in this simulation are all averages over a number of trials.

Fig. 6 shows the average message delivery time, LTMD$_{time}$, for the two protocols, MFPP and MAGCP, for the specified range of the Stay$_{time}$ in case MAGroupSize$_{no}$ is 200 and Set$_{snd}$ is 3. In this figure, as their Stay$_{time}$s decrease, their LTMD$_{time}$s are contrarily becoming longer. This phenomenon results from the reason that short mobile agent's stay period has length of the path for forwarding messages to the agent become longer. However, this figure indicates our protocol MAGCP reduces about 11.8-54.6$\%$ of LTMD$_{time}$ compared with MFPP. The first reason is that MAGCP allows each agent to choose the smaller number of message forwarders among all of its visiting nodes unlike MFPP. The second reason is that sender's mobile agent group location cache of MAGCP may significantly accelerate the speed of message delivery to each agent member. In particular, as Stay$_{time}$ is shorter, this figure demonstrates that higher performance improvement in the message delivery time may be obtained from these desirable features of MAGCP.

Fig. 7 shows the amount of agent location information kept by each service node, LInfo$_{no}$, for the two protocols with varying MAGroupSize$_{no}$ in case Stay$_{time}$ is 4 seconds and Set$_{snd}$ is 3. As MAGroupSize$_{no}$ becomes bigger, their LInfo$_{no}$s are also increasing. This outcome arises from the reason that the increase of the size of mobile agent group leads to the larger number of agent

location entries each node should keep on its storage. But, this simulation results indicate MAGCP reduces about 38-52% of LInfo_{no} compared with MFPP. Especially, as MAGroupSize_{no} grows larger, the increasing rate of MAGCP's LInfo_{no} is stepping up at a much lower speed compared with MFPP's.

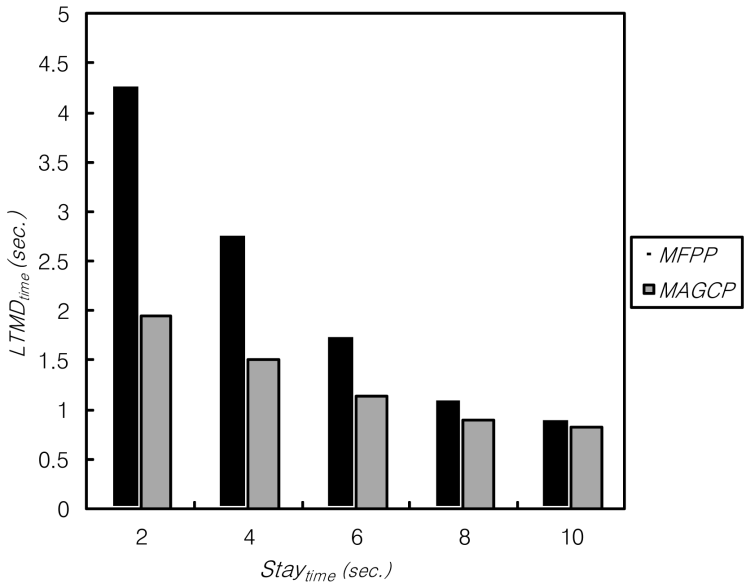

Fig. 6. Message delivery time

7. Conclusion

This paper proposes an atomic mobile agent group communication protocol to achieve all the following requirements existing protocols couldn't address due to their respective limitations. This protocol improves scalability by enabling each mobile agent to choose only a few among its visiting nodes as agent location manager depending on its preferred policies such as location updating and message delivery costs, security, network latency and topology, inter-agent communication patterns, etc.. Second, to guarantee agent communication reliability despite agent location managers's failures, it allows each mobile agent's location information to be replicated in an effective way to preserve its scalability to a maximum. Also, it has messages destined to an agent group to be reliably delivered to its surviving group members in the same order. Lastly, each sending agent's agent group location cache significantly allows message delivery

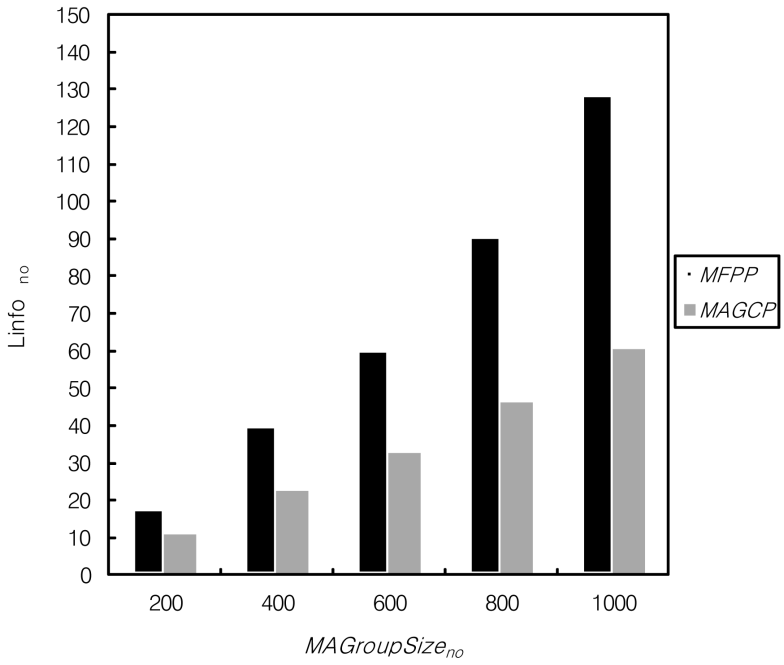

Fig. 7. Location management overhead

time to the targeted mobile agents to be shortened and message forwarding load imposing on agent location managers to decrease.

Acknowledgments. This work was supported by Kyonggi University Research Grant 2010(Project Number: 2010-003).

References

1. Ahn, J.: Lightweight Fault-tolerance Mechanism for Distributed Mobile Agent-based Monitoring. In Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference. IEEE Press, Las Vegas, NV, USA, 1232-1236. (2009)
2. Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Zeng, X., Martin, J., Song, H.: Parsec: A Parallel Simulation Environments for Complex Systems. IEEE Computer, Vol. 31, No. 10, 77-85. (1998)
3. Cao, J., Feng, X., Lu, J., Das, S.: Mailbox-based scheme for mobile agent communications. IEEE Computer, Vol. 35, No. 9, 54-60. (2002)
4. Cimino, M., Marcelloni, F.: Autonomic tracing of production processes with mobile and agent-based computing. Information Sciences, Vol. 181, No. 5, 935-953. (2011)

5. Cooper, C., Klasing, R., Radzik, T.: Locating and repairing faults in a network with mobile agents. Theoretical Computer Science, Vol. 411, No. 14-15, 1638-1647. (2010)
6. Defago, X., Schiper, A., Urban., P.: Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. ACM Computing Surveys, Vol. 36, No. 4, 372-421. (2004)
7. Elnozahy, E. N., Alvisi, L., Wang, Y. M., Johnson, D. B.: A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Computing Surveys, Vol. 34, No. 3, 375-408. (2002)
8. Lin, K., Chen, M., Zeadally, S., Rodrigues, J.: Balancing energy consumption with mobile agents in wireless sensor networks. Future Generation Computer Systems, Vol. 28, No. 2, 446-456. (2012)
9. Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., Kosaka, K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S., White, J.: MASIF: The OMG Mobile Agent System Interoperability Facility. In: Rothermel, K., Hohl, F. (eds.): Mobile Agents. Lecture Notes in Computer Science, Vol. 1477. Springer-Verlag, Berlin Heidelberg New York, 50-67. (1998)
10. Moreau, L.: Distributed Directory Service and Message Router for Mobile Agents. Science of Computer Programming, Vol. 39, No. 2-3, 249-272. (2001)
11. Moreau, L.; A Fault-Tolerant Directory Service for Mobile Agents based on Forwarding Pointers. In Proceedings of the 17th ACM Symposium on Applied Computing. ACM New York, Madrid, Spain, 93-100. (2002)
12. Murphy, A. L., and Picco, G. P.: Reliable Communication for Highly Mobile Agents. Journal of Autonomous Agents and Multi-Agent Systems, Vol. 5, No. 1, 81-100. (2002)
13. Perkins, C.: IP Mobility Support for IPv4, RFC 3344, (2002)
14. Schlichting, R. D., Schneider, F. B.: Fail-stop processors: an approach to designing fault-tolerant distributed computing systems. ACM Transactions on Computer Systems, Vol. 1, No. 3, 222-238. (1985)
15. Schmidt, T. C., Wahlisch, M., Fairhurst, G.: Multicast Mobility in MIPv6: Problem Statement and Brief Survey. IRTF Internet Draft, No. 08, (2009).
16. Su, C., Wu, C.: JADE implemented mobile multi-agent based, distributed information platform for pervasive healthcare monitoring. Applied Soft Computing, Vol. 11, No. 1, 315-325. (2011)

Jinho Ahn received his B.S., M.S. and Ph.D. degrees in Computer Science and Engineering from Korea University, Republic of Korea, in 1997, 1999 and 2003, respectively. He has been an associate professor in Department of Computer Science, Kyonggi University since 2003. He has published more than 70 papers in refereed journals and conference proceedings and served as program or organizing committee member or session chair in several domestic/international conferences and editor-in-chief of journal of Korean Institute of Information Technology and editorial board member of journal of Korean Society for Internet Information. His research interests include distributed computing, fault-tolerance, sensor networks and mobile agent systems.

Received: July 16, 2012; Accepted: December 6, 2012.